

TP2 MI11/AI39 Linux Xenomai

Semestre printemps 2025

Guillaume Sanahuja – guillaume.sanahuja@hds.utc.fr

Table des matières

Préambule.....	1
Travail préalable à la séance de TP.....	1
Exercice : Pathfinder.....	2

Préambule

Pour ce TP, vous devez réutiliser la machine virtuelle de la séance précédente.

Un compte rendu de TP au format PDF est à rendre sur le moodle de l'UV. Il vous est demandé d'y écrire les réponses aux questions figurant dans les différents exercices. Également, vous pouvez compléter le compte rendu avec toute information que vous jugerez utile, par exemple les manipulations que vous avez réalisées pendant la séance, les résultats que vous avez observés, etc.

Travail préalable à la séance de TP

Relire les cours sur Linux embarqué et sur Linux temps réel disponibles sur le moodle MI11. Regarder la documentation de Xenomai :

<https://doc.xenomai.org/v3/html/xeno3prm/index.html>

Et lire notamment la documentation sur l'API alchemy, que vous utiliserez pour ce TP :

https://doc.xenomai.org/v3/html/xeno3prm/group_alchemy.html

Exercice : Pathfinder

Nous allons nous intéresser dans cet exercice au programme informatique de la mission martienne Pathfinder. Ce projet fut un véritable challenge car mené en seulement 3 ans pour un budget de 125 millions de dollars (la norme étant alors autour de 10 ans pour un budget d'un milliard de dollars). Le gros défi de la mission résidant dans le système d'atterrissage avec des coussins gonflables. D'autres aspects de la mission ont alors (peut être) été plus négligés. Ainsi, après quelques temps sur Mars, la NASA s'est rendue compte que le système informatique redémarrait de façon intempestive. Les ingénieurs avaient bien observé quelques fois ce genre de phénomène avant le lancement, mais il avait alors été considéré que sa faible occurrence ne serait pas un problème. Cet exercice va vous permettre de reproduire ce phénomène, et d'y trouver une solution. Le problème sera ici largement simplifié et adapté, car le but est purement pédagogique.

L'architecture matérielle de la sonde se compose d'un CPU connecté à une carte radio, à une carte caméra et à un bus 1553 (pour les autres capteurs et actionneurs).

Le système temps réel embarqué est VxWorks, permettant de faire tourner un certain nombre de tâches. Celles que nous étudierons sont les suivantes :

Tâche	Priorité	Fonctionnalité	Période	Temps d'exécution	Remarque
ORDO_BUS	7	Ordonnanceur du bus 1553	125 ms	25 ms	
DISTRIB_DONNEES	6	Distribution des données du bus 1553	125 ms	25 ms	Utilise le bus 1553
PILOTAGE	5	Pilotage du robot	250 ms	25 ms	Utilise le bus 1553
RADIO	4	Gestion des communications radios	250 ms	25 ms	
CAMERA	3	Gestion de la caméra	250 ms	25 ms	
MESURES	2	Gestion des mesures diverses	5000 ms	50 ms	Utilise le bus 1553
METEO	1	Gestion des données météo	5000 ms	40 à 60 ms	Utilise le bus 1553

Deux tâches gèrent le bus 1553 : DISTRIB_DONNEES qui distribue les données sur le bus, et ORDO_BUS qui en fait l'ordonnancement et vérifie que DISTRIB_DONNEES fonctionne correctement. ORDO_BUS n'accède pas directement au bus. Les autres tâches utilisant le bus devront par contre être synchronisées pour éviter tout problème car elles accèdent à la même ressource.

Vous trouverez sur le moodle de l'UV le squelette du programme (fichier *pathfinder.c*) que vous complèterez au fil de l'exercice, ainsi que le *Makefile* associé.

Question 1 : Expliquez le principe des fonctions `create_and_start_rt_task`, `rt_task` ainsi que de la structure `task_descriptor`.

Question 2 : Expliquez l'utilité de la variable `first_release_point` et du sémaphore `start_sem`.

Question 3 : Donnez, en le justifiant, les différentes étapes à réaliser dans le main afin de créer et synchroniser le démarrage de plusieurs tâches temps réel périodiques.

Complétez le *main* afin de créer et lancer la tâche `ORDO_BUS`.

Question 4 : Expliquez le principe de la fonction `rt_task_name`. Quelles autres informations sont stockées dans la structure `RT_TASK_INFO`? Cherchez également la définition de la structure `threadobj_stat`.

Les différentes tâches vont avoir besoin de la fonction `busy_wait` qui permet de simuler le temps d'exécution d'une tâche. Cette fonction devra donc faire une attente active (et non pas un simple `sleep`), tant que la tâche ne s'est pas déroulée pendant la durée prévue. Le champ `xtime` de la structure `threadobj_stat` vous permettra d'obtenir le temps total passé dans une tâche.

Question 5 : Décrivez comment vous avez réalisé la fonction `busy_wait`.

Utilisez la fonction `ms_time_since_start` pour afficher et vérifier les temps dans la fonction `rt_task` puis testez le fonctionnement de votre programme.

Question 6 : Quel est le résultat? Le timing est-il correct?

Ajoutez maintenant les autres tâches. Afin de gérer l'accès au bus 1553, créez un nouveau sémaphore qui sera utilisé par les fonctions `acquire_resource` et `release_resource`. Testez le fonctionnement de votre programme.

Question 7 : Expliquez comment le sémaphore doit être initialisé. Vérifiez l'enchaînement des tâches.

Afin d'éviter tout blocage du CPU à cause d'un bug, les ingénieurs ont ajouté plusieurs sécurités dans le programme. En cas de plantage, le système doit redémarrer automatiquement; ce qui évite d'envoyer un homme sur Mars pour appuyer sur le bouton `reset`... La sécurité nous intéressant dans cet exercice est celle implémentée dans la tâche `ORDO_BUS` qui vérifie que `DISTRIB_DONNEES` fonctionne correctement. Entre deux exécutions de `ORDO_BUS`, `DISTRIB_DONNEES` doit s'être exécutée entièrement.

Créez donc deux nouvelles fonctions pour les tâches `ORDO_BUS` et `DISTRIB_DONNEES` afin de ne plus utiliser la fonction par défaut (`rt_task`). Pour implémenter le mécanisme de sécurité, utilisez un nouveau sémaphore.

Question 8 : Expliquez le fonctionnement de la solution retenue.

Ajoutez le message `reset` en cas de dysfonctionnement et faites terminer le programme.

Question 9 : Testez votre programme pour les cas extrêmes du temps d'exécution de `METEO`. Qu'observez-vous? Expliquez ce phénomène à l'aide de chronogrammes.

Question 10 : Quelle solution proposez-vous pour résoudre le problème?

Implémentez cette solution, de façon à ce que chaque version (fonctionnelle et non fonctionnelle) soit présente dans votre code. Le choix pouvant se faire à la compilation à l'aide de `#ifdef`.

Question 11 : Testez et commentez le résultat. Montrez l'impact de votre solution à l'aide de chronogrammes.

Question 12 : Fournissez le code complet du programme, en prenant soin de le commenter.